MORRISON & FOERSTER LLP
MICHAEL A. JACOBS (Bar No. 111664)
mjacobs@mofo.com
KENNETH A. KUWAYTI (Bar No. 145384)
kkuwayti@mofo.com
MARC DAVID PETERS (Bar No. 211725)
mdpeters@mofo.com
DANIEL P. MUINO (Bar No. 209624)
dmuino@mofo.com
755 Page Mill Road, Palo Alto, CA 94304-1018
Telephone: (650) 813-5600 / Facsimile: (650) 494-0792

BOIES, SCHILLER & FLEXNER LLP
DAVID BOIES (Admitted *Pro Hac Vice*)
dboies@bsfllp.com
333 Main Street, Armonk, NY 10504
Telephone: (914) 749-8200 / Facsimile: (914) 749-8300
STEVEN C. HOLTZMAN (Bar No. 144177)
sholtzman@bsfllp.com
1999 Harrison St., Suite 900, Oakland, CA 94612
Telephone: (510) 874-1000 / Facsimile: (510) 874-1460

ORACLE CORPORATION
DORIAN DALEY (Bar No. 129049)
dorian.daley@oracle.com
DEBORAH K. MILLER (Bar No. 95527)
deborah.miller@oracle.com
MATTHEW M. SARBORARIA (Bar No. 211600)
matthew.sarboraria@oracle.com
500 Oracle Parkway, Redwood City, CA 94065
Telephone: (650) 506-5200 / Facsimile: (650) 506-7114

Attorneys for Plaintiff
ORACLE AMERICA, INC.

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

ORACLE AMERICA, INC. Plaintiff, v. GOOGLE INC. Defendant.	Case No. CV 10-03561 WHA **ORACLE MAY 23, 2012 COPYRIGHT BRIEF** Dept.: Courtroom 8, 19th Floor Judge: Honorable William H. Alsup

Oracle submits this brief in response to the Court's request for further briefing regarding interfaces, exceptions and interoperability (ECF No. 1181).

I. INTERFACES AND EXCEPTIONS

A. Please state how many "interfaces" are included in the joint table supplied by counsel. (Dkt. No. 1124). Also please state how many "exceptions" were "thrown" and the extent to which they were duplicated by Google. Each side should also include one example of an "interface" and one example of a "throw" to illustrate the most salient points about these features.

1. Interfaces

The joint table at ECF No. 1124 includes 171 interfaces from Java 2 Standard Edition ("J2SE") 5.0 and 158 interfaces from Android Froyo.

Interfaces can be used to group classes in different packages. For example, the interface java.util.Set is implemented by seven classes spread across three different packages. *See* TX 610.2 at /docs/api/java/util/Set.html under "All Known Implementing Classes." The classes implementing java.util.Set include, among others, java.util.EnumSet, java.util.HashSet, java.util.concurrent.CopyOnWriteArraySet, and javax.print.attribute.standard.JobStateReasons. *Id.*

While a class can only inherit from one superclass, it can implement more than one interface. TX 984 at 259. The class java.util.HashSet, for example, implements the interfaces java.util.Set, java.lang.Cloneable and java.io.Serializable. TX 610.2 at /docs/api/java/util/HashSet.html. An interface establishes relationships that might not otherwise exist between classes. *See* TX 984 at 259; RT 589:13-590:23 (Reinhold); RT 1239:5-7 (Mitchell).

Interfaces influence SSO at the method level as well. Every method declared in an interface must be implemented by each class that implements that interface, or else be inherited from a superclass of that class. TX 984 at 224. As mentioned above, java.util.HashSet implements the interface java.util.Set, which declares 15 distinct methods. The HashSet class implements eight of those methods itself (add, clear, clone, contains, isEmpty, iterator, remove and size); it inherits three of them (equals, hashCode, removeAll, and the toArray methods) from

its direct superclass AbstractSet; and it inherits the remainder from AbstractCollection, the direct superclass of AbstractSet. TX 610.2 at /docs/api/java/util/HashSet.html.

Interfaces can also be subtypes of interfaces in different packages. RT 601:22-602:1 (Reinhold) (interfaces in java.nio.Channels are subtypes of the interface java.io.Closeable).

Google copied many of the classes that implement the interfaces defined in the J2SE API, but not all of them. For example, Google implemented only a subset of the classes that implement the interface java.util.Set: Android does not include the class javax.print.attribute.standard.JobStateReasons, which is present in J2SE, because Android did not include the javax.print package. *Compare* TX 610.2 at /docs/api/java/util/Set.html *with* TX 767 at /java/util/Set.html. As the above shows, the interfaces in J2SE embody significant creative expression. Google copied them, including their complex SSO, by choice, not by necessity.

2. Exceptions

Exceptions are used to report error conditions. "When a program violates the semantic constraints of the Java programming language, the Java virtual machine signals this error to the program as an exception." "Programs can also throw exceptions explicitly, using throw statements." TX 984 at 297. The API documentation for J2SE 5.0 specifies 176 exceptions in the 37 asserted packages. *See* package listings for each of the 37 J2SE API packages in TX 610.2. Of these, 165 are copied into Android Froyo. *Compare id. with* TX 767.

Based on an analysis of the compiled class libraries for J2SE 5, the throws clauses (exception lists) of the 6,508 J2SE 5.0 methods in the table at ECF No. 1142 contain 2,220 exceptions (including scenarios when multiple methods threw the same exceptions). Based on a similar analysis, there are 2,241 exceptions mentioned in the exception lists for Android Froyo. Of these, 1,828 throws clauses (comprising 2,014 exceptions) are *identical* between J2SE 5.0 and Android Froyo, including the order of exceptions.

The list of exceptions a method throws represents the expression of a structural choice. According to the *Java Language Specification*, "When an exception is thrown, control is transferred from the code that caused the exception to the nearest dynamically-enclosing catch clause of a try statement (§14.20) that handles the exception." TX 984 at 302. The "catch"

clause that handles the exception can be within the method either directly or indirectly. *See id.* For example, the method java.security.cert.Certificate declares the following method:

> *public abstract void verify(PublicKey key, String sigProvider)*
> *throws CertificateException, NoSuchAlgorithmException, InvalidKeyException, NoSuchProviderException, SignatureException*

TX 610.2 at /docs/api/java/security/cert/Certificate.html. This method "[v]erifies that this certificate was signed using the private key that corresponds to the specified public key," and it throws various exceptions in the event of unsupported signature algorithms, an incorrect key, and other scenarios. *Id.* By specifying that this method "throws" five different types of exceptions, the API designers have required any code that calls the method to handle all five error scenarios.

In Android, Google could have created different exception lists using supertypes of the listed exceptions. "For each checked exception that can result from execution of the body of a method or constructor, a compile-time error occurs unless that exception type *or a supertype* of that exception type is mentioned in a throws clause in the declaration of the method or constructor." TX 984 at 221 (emphasis added). For example, in the case of the verify() method listed above, all of the listed exceptions are subtypes of java.security.GeneralSecurityException. TX 610.2 at /docs/api/java/security/GeneralSecurityException.html. The method would still compile if the throws clause had included only GeneralSecurityException.

The *Java Language Specification* places no significance on the order of the exceptions in a throws clause. *See* TX 984 at 221-22. The fact that Android copied 1,828 exception lists verbatim shows that Google copied more than what it required for its alleged goal of "compatibility." To the extent that the exception lists are identical between Java and Android, it is because the Android API designers *chose* to make them identical.

II. EVIDENCE REGARDING INTEROPERABILITY

A. To what extent, if at all, have applications and programs written for the J2SE platform before Android arrived been able to run on Android?

No application or program written for the J2SE platform runs on Android. There are several reasons for this: (a) Android applications start up in a completely different way from standard Java applications; (b) Android applications are compiled to Dalvik bytecode rather than

Java bytecode; and (c) Android applications are packaged as "apk" files, which are similar to but incompatible with Java's "jar" files. Many applications and programs written for the J2SE platform will not run on Android for the additional reason that Google did not include all of the Java API packages and classes. Evidentiary support for each of these points can be found in the trial record as described below.

The Q&A that Google developed for the Android announcement and "early look" release of the Android SDK in November 2007 confirms that Android is not interoperable with Java:

> Q48. Does Android support existing Java apps?
>
> A. No.
>
> Q49. Is Android Java compatible?
>
> A. No.

TX 383 at 8.

No application written for the Java platform will run on Android, even if it uses only the 51 API packages from J2SE 5.0 that Google copied into Android. Java platform applications are required to include a special method called "main" that serves as the entry point (where program execution begins), but Android does not use the main() entry point. The Java platform requirement for the main() method may be found, for example, in Section 2.16.1 of the Java Virtual Machine Specification:

> A Java Virtual Machine starts execution by invoking the method main of some specified class, passing it a single argument, which is an array of strings. This causes the specified class to be loaded (§2.16.2), linked (§2.16.3) to other types that it uses, and initialized (§2.16.4). The method main must be declared public, static, and void.

TX 25 at 40; *see also* TX 984 at 309.

Android embodies a different application model than the Java platform. It does not use the main() method as the entry point of an application. As Dr. Astrachan confirmed, Java programs must be rewritten to run on Android:

> Q. Mr. Jacobs asked you about the program that you wrote this morning, and he asked you whether it would run, whether it would compile on the Android platform as opposed to Java. Would you have written this program any differently if someone told you it needed to run on the Android platform as opposed to the Java Platform?

> A. I would have changed the main. That's a Java entry point. Otherwise nothing else would change.
>
> Q. Okay.
>
> THE COURT: Change what?
>
> THE WITNESS: The main.
>
> THE COURT: Where is that?
>
> THE WITNESS: The Public Static Void Main. That's a requirement of the Java Platform and the Java Language on that platform. That's the launch point for a program.
>
> THE COURT: You would change it to what?
>
> THE WITNESS: For how to do that on Android. That's a little different.

RT 2221:11-2222:3 (Astrachan).

Java applications will not run on Android for the additional reason that Java applications are compiled to Java bytecode, while Android applications are compiled to Dalvik bytecode. *See* RT 2287:9-22 (Mitchell). Since programs are not normally distributed in source code format for security and efficiency reasons, this means a program compiled for distribution for the Java platform has to be recompiled by the developer for distribution for the Android platform before it can run on Android. An Android phone does not have the ability to convert Java class files to Android dex files.

A third reason why Java applications will not run on Android is that Java and Android use different file formats. Java applications are distributed in "jar" files. *See, e.g.*, TX 610.2 at /docs/guide/jar/jar.html. Google distributes Android applications in "apk" files. *See* TX 757.

This is only the tip of the iceberg, however. Oracle has already described elsewhere how many categories of Java applications that will not run on Android because Google chose not to include the API packages or classes they require for common functions like interacting with the graphical user interface, sound, image input/output and printing. *See* ECF No. 1118 at 18-19. While Oracle is not aware of any metric that measures the exact percentage of programs that are incompatible for this reason, it is instructive to look at the relatively simple applications Sun distributed with JDK5 as demos and samples to teach developers how to program for the Java

platform. These demos and samples are contained in the JDK5 archives found in the directory licenseebundles/jdk1.5.0 of TX 623. There are 36 relevant examples. Of these only one, the sample application in sample/nio/server, might run on Android if compiled into Dalvik bytecode and distributed as an apk file but for the fact that, like all Java applications, it uses main() as an entry point. *See* RT 2221:11-2222:3 (Astrachan). The 20 Java demo applets in demo/applets will not run on Android because Android does not support applets and is missing the required java.applet and java.awt API packages and subpackages. *See* TX 1072 (list of copied packages). The 11 demo applications in demo/jfc will not run in Android because Android is missing the required java.awt and javax.swing API packages and subpackages, among other reasons. *See id.* The four applications in demo/management will not run on Android because Android is missing the required java.lang.management API package, among other reasons. *See id.*[1]

The fact that Android will not run simple applications designed to demonstrate the features of the Java platform is strong evidence that Android was never meant to be compatible with Java.

B. To what extent, if at all, have applications and programs written after Android arrived been able to run both on Android and J2SE?

As shown above, applications written for the Java platform do not run on Android. This did not change after Android arrived. Applications written for Android do not run on the Java platform. As with the JDK5 examples above, an examination of the sample applications that Google provides with the Android SDK to teach developers how to program for Android shows that none of the Android applications Google included will run on the Java platform, even though they are written in the Java programming language. (*See* TX 43\cupcake15 - GOOGLE-00-00000523\development\samples.) Dr. Bloch testified that the "simplest program that you can write in any language is called the Hello World Program. It just uses that programming language to print Hello World, and that's how you sort of start off with [a] new language. You learn how to

[1] The directory demo/jvmti also contains seven "instrumentation agents" that demonstrate how the Java Virtual Machine Tools Interface (JVMTI) can be used to access information in a running VM, but these will not work on Android since Android doesn't include JVMTI. However JVMTI is optional under the TCK rules for J2SE 5.0, so Oracle does not include them in its count.

write a program that simply prints Hello World in the language." RT 782:23-783:2 (Bloch). Google's "Hello World" application for Android is called "HelloActivity.java," and it will not run on the Java platform. *See* HelloActivity.java in TX43\cupcake15 - GOOGLE-00-00000523\development\samples\HelloActivity\src\com\example\android\helloactivity. Google's HelloActivity.java lacks a main() method, which is a required Java platform entry point, and it requires classes and methods defined in the android.app and android.os API packages, which do not exist in the Java platform. *Id.*; *see* TX 1072 (Android API packages copied from Java begin with "java" or "javax."); s*ee also* RT 2180:3-15 (Astrachan).

The other sample applications, which are more complex than "Hello World" yet still relatively simple, are not interoperable with Java either. For example, the LunarLander application requires classes and methods defined in the android.app, android.content, android.content.res, android.graphics, android.graphics.drawable, android.os, android.util, android.view, android.widget API packages, none of which exist in the Java platform. *See* LunarLander.java and LunarView.java in TX43\cupcake15 - GOOGLE-00-00000523\development\samples\LunarLander\src\com\example\android\lunarlander; TX 1072. Similarly, none of the standard "stock" applications for Android, which include the calculator, calendar, and alarm clock, are interoperable with Java. *See, e.g.*, AlarmClock.java in TX43\cupcake15 - GOOGLE-00-00000523\packages\apps\AlarmClock\src\com\android\alarmclock; TX 1072.

When even Google's simple, sample applications will not run on the Java platform, it confirms what Google said at the inception of Android: Android does not support existing Java applications, and Android is not Java-compatible. TX 383 at 8.

C. How, if at all, have Android and the replication of virtually all of the 37 packages promoted interoperability?

Android's use of the 37 API packages copied from Java has not promoted interoperability. Android has harmed interoperability.

As discussed in sections II.A and II.B above, Android and the Java platform are not interoperable. Applications and programs written for one will not run on the other. *See*

RT 1331:16-1332:2 (Mitchell) ("So you don't really have compatibility. You can't ship code from one platform to another.").

Google will no doubt argue that by copying some of the Java APIs packages it has furthered interoperability to some degree. But Google cannot claim interoperability even for the 37 APIs Google it copied nearly identically because it did not copy *everything* from those packages. *See* ECF No. 1124-1 at Ex. A (missing classes and methods in java.awt.font, java.beans, javax.security.auth, java.security.auth.callback, javax.security.auth.login, and javax.security.auth.x500). The result is, as noted above, that many standard Android applications, including "the simplest program you can write in any language" (RT 782:23-783:2 (Bloch)), cannot run on Java. *See* section II.B *supra*.

This type of partial, selective copying is harmful to interoperability, not helpful. Google's strategy, known as "embrace and extend," was to adopt enough of the Java APIs to attract Java developers to Android, but not enough to make Android compatible. Sun successfully raised a similar claim in the *Sun v. Microsoft* case, where it alleged Microsoft "embraced" Java by licensing the Java technology and then "extended" it by developing "strategic incompatibilities" in the version of Java it created for the Windows platform. *Sun Microsystems, Inc. v. Microsoft Corp.*, 87 F. Supp. 2d 992, 995 (N.D. Cal. 2000). Judge Whyte recognized how damaging such a partially compatible platform can be:

> In the present case, Sun has demonstrated a possibility of irreparable harm, if an injunction restraining Microsoft's distribution of non-compliant Java Technology is not issued. Microsoft's unauthorized distribution of incompatible implementations of Sun's Java Technology threatens to undermine Sun's goal of cross-platform and cross-implementation compatibility. The threatened fragmentation of the Java programming environment harms Sun's relationship with other licensees who have implemented Java virtual machines for Win32-based and other platforms. In addition, Microsoft's unparalleled market power and distribution channels relating to computer operating systems pose a significant risk that an incompatible and unauthorized version of the Java Technology will become the *de facto* standard. The court further finds that money damages are inadequate to compensate Sun for the harm resulting from Microsoft's distribution of software products incorporating non-compliant Java Technology as the harm to Sun's revenues and reputation is difficult to quantify.

Id. at 997-98. *See also ADA v. Delta Dental Plans Ass'n*, 126 F.3d 977, 981 (7th Cir. 1997) (noting that "standardization of language promotes interchange among professionals" and "[t]he fact that Delta uses most of the Code but made modifications is the reason ADA objects").

Google's distribution of its "free" Android platform is just as harmful to Oracle. Oracle is forced to compete using its for-charge licensing model against an infringing system that is licensed for free. And, similar to Microsoft in the operating system market, Google has achieved a dominant position in the smartphone market, where Android phones containing these incompatible APIs are being activated at a rate of 750,000 phones per day. *See* RT 1017:4-16 (Morrill).

If Google wanted to promote interoperability, it could have made Android compatible. A well-developed system is in place for promoting the interoperability of Java implementations and the platform's "write once, run anywhere capability," supported by many individuals and companies. *See, e.g.,* RT 293:8-296:4 (Ellison); 360:6-363: 10 (Kurian); 2055:7-21 (McNealy). The Java specification license allows an independent implementation if, among other things, it "fully implements" the specification, does not "modify, subset, superset or otherwise extend" the Java name space, and passes the TCK. TX 610.1. Google itself acknowledged that "[t]he only way to demonstrate compatibility with the Java specification is by meeting all of the requirements of Sun's Technology Compatibility Kit, TCK, for a particular edition of Sun's Java." RT 976:16-978:1 (deemed admission). But Google never even attempted to have Android pass the TCK. (RT at 984:22-24 (Lee).)

Rather than further interoperability by following these procedures and taking a license, Google undermined interoperability by creating an incompatible fork of the Java platform. The uncontroverted evidence at trial was that Google is the only company that is commercially using the Java APIs that has not taken a license. *See, e.g.,* RT 293:8-294:21 (Ellison); 385:20-386:8 (Kurian), 487:10-488:7 (Page). Google is a source of incompatibility, not interoperability.

D. To what extent was interoperability an actual motive of Google at the time the decision was made to replicate the 37 packages?

Google never proved interoperability was its motive in replicating the 37 packages. To the contrary, the evidence showed Google did not care about interoperability. It copied what it wanted to provide Java developers with a familiar enough programming environment so they would migrate over to Android.

Under questioning from his own counsel, Google developer Dan Bornstein testified bluntly that interoperability was not one of Google's goals:

> Q. Did Android implement all the API packages present in any particular Java Platform?
>
> A. No.
>
> Q. All right. And why not?
>
> A. That wasn't a goal of the project. The goal of the project was to provide something that was familiar to developers. It wasn't to provide any particular preexisting set of packages.

RT at 1783:15-22 (Bornstein).

Daniel Morrill, the technical program manager for Android compatibility, testified at trial that Google has a compatibility program that "is intended to make sure that compatible Android devices can run applications written to the Android SDK." (1001:10-12, 1009:19-1010:4). But he made it clear Google does not even try to support applications written to the Java SDK:

> Q. Now, Android does not support Java applications, correct?
>
> A. That is correct.
>
> Q. And so Android is not Java compatible, correct?
>
> A. That's correct.

RT 1010:4-7 (Morrill).

Even Google's counsel conceded that, rather than promote interoperability, Google chose to replace Java API packages with Android packages that Google thought were "better:"

> So there were lots of packages that would make no sense to put them in Android, the user interface, some of the other things that make a smart phone a smart phone. So there was no reason to put those in Android.

> *Then there were others where we were doing our own Android specific APIs, so there was no need for the Java ones. We had better ones that we wanted to put in for our APIs.*
>
> So you started with all of them. You took out the ones that weren't applicable. *You took out the ones where we had better ones.* And what you had left was this collection of 51 that were certainly all of the core ones.

RT 1138:21-1139:12 (emphasis added).

Google's approach stands in contrast to what Andy Rubin claimed Danger did when it created its smartphone platform. Mr. Rubin stated that Danger expressly set out to try to be interoperable with Java. *See* RT at 1588:19-25 ("Well we wanted to make sure that when somebody, when one of those university students who graduated knew how to program in Java wrote a program, we wanted to make sure that that program could run on other devices too, not only our device. So we wanted to make sure that we were compatible with other devices that happened to be running the Java programming language."). Mr. Rubin did not claim there was any such goal for Android.

The most Google has argued was that it was trying to create some kind of limited interoperability for the 37 API packages it copied. Dr. Astrachan made this argument. *See* RT 2183:2-11, 2224:3-8 (Astrachan). Dr. Astrachan, of course, cannot speak directly to Google's motive, which is not a proper subject of expert testimony. The actual evidence that Google intended to further even this limited form of interoperability was scant. Mr. Lee mentioned it in passing, claiming Google referred back to Sun's API specifications "to make sure that we were maintaining—not—or maintaining interoperability with their implementations." RT 1201:12-22 (Lee). Mr. Lee certainly did not claim Google was trying to achieve interoperability overall, however. Instead, he testified that Google only "supports certain Java APIs," which he described as the "good stuff from Java." TX 1067 (Lee Dep.) at 48:10-14, 48:16 (played at RT 982:15-21).

In any event, neither Mr. Lee's nor Dr. Astrachan's testimony supports the notion that Google was trying to achieve interoperability even with the 37 API packages at issue because, as noted in section II.C above, the parties agree that Google did not include all of the classes or interfaces from even those 37 APIs. *See* ECF No. 1124.

Moreover, even if Google had what it thought were "better" APIs, if its motive had been to promote interoperability then in cases where Google had alternate APIs it preferred it still could have supported the use of the existing standard Java user-interface APIs alongside its own. Google presented no evidence at trial that it ever did anything of the kind.

This case is not *Sony* or *Sega* for many reasons. *See, e.g.,* ECF No. 853 at 15-16. Of particular relevance here, however, is that Google's copying has nothing to do with the type of interoperability addressed in those cases. In *Sony*, the defendant engaged in reverse engineering that was necessary to develop a final product which Sony did not allege infringed its copyrights. *Sony Computer Entm't, Inc. v. Connectix Corp.*, 203 F.3d 596, 600, 603 (9th Cir. 2000). Likewise in *Sega*, the court found reverse engineering to be fair use "[w]here disassembly is the only way to gain access to the ideas and functional elements embodied in a copyrighted computer program and where there is a legitimate reason for seeking such access[.]" *Sega Enters. Ltd. V. Accolade, Inc.*, 977 F.2d, 1510, 1527-28 (9th Cir. 1992). The court emphasized that its decision "does not, of course, insulate Accolade from a claim of copyright infringement with respect to its finished products." *Id.* at 1528. Notably, in rejecting the contention that copying Sega's simple 20 byte initialization code was not fair use, the court distinguished this basic sequence from the console key in *Atari v. Nintendo*, because "Creativity and originality went into the design of that program." *Id.* at 1524 n. 7 (citing *Atari v. Nintendo*, 975 F.2d 832, 840 (Fed. Cir. 1992).)

Here, Google's copying was neither "intermediate" nor necessary. Google's expert admitted that Google could have designed its own APIs and in fact did so elsewhere. *See, e.g.,* RT 2212:25-2213:19 (Astrachan). Dan Bornstein testified that Google's copying included API packages "where it might—it might not be necessary, but it would be surprising to not find them." RT 1782:6-1783:10 (Bornstein). Copying to lure a competitor's developers is not the same as reverse engineering to ensure interoperability. Unlike the Accolade games that would not have functioned on Sega's Nintendo system unless Accolade copied the 20 byte instruction sequence, Android could have functioned perfectly well if Google had not copied the Java APIs that took more than a decade to build. And unlike *Sony* and *Sega*, Android is not interoperable with Java. The reasoning in *Sony* and *Sega* does not apply here.

Dated: May 23, 2012

MORRISON & FOERSTER LLP

By: */s/ Michael A. Jacobs*
Michael A. Jacobs

Attorneys for Plaintiff
ORACLE AMERICA, INC.